QAnvas



Q & A Chatbot for Canvas

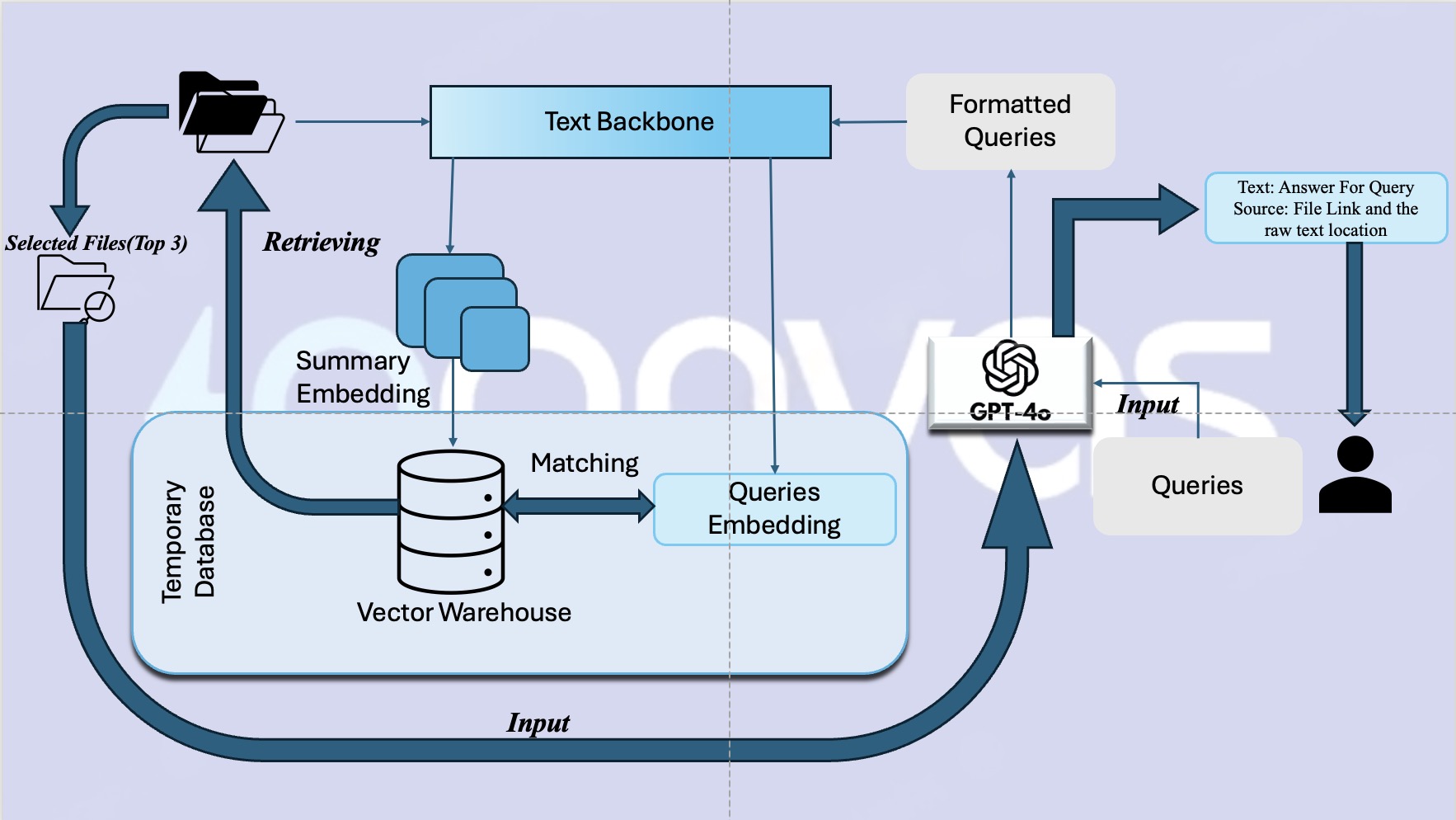
Liu Mohan, Ding Yi, Zhou Yuhao, Ng Min Teck

# 1. Overview

The **proliferation of online learning platforms** like Canvas has revolutionized education by providing seamless access to course materials, assignments, and communication tools for students and instructors. However, the **vast amount of data** on these platforms—encompassing course schedules, announcements, assignments, discussions, grades, and resources—often overwhelms users, making it difficult to efficiently locate relevant information or obtain personalized support. This **information overload** results in wasted time, missed deadlines, and decreased productivity, presenting a **critical business challenge** for educational institutions and learners alike.

This project proposes the development of an advanced **Language Model (LLM)-powered chatbot** integrated with Canvas to address these challenges. The chatbot will harness **natural language processing** to enable intuitive queries (e.g., "What are my upcoming assignments?" or "What did my instructor announce today?"), delivering **real-time, accurate responses**. By streamlining information retrieval and enhancing user experience, the solution will improve **time management** and boost engagement within the Canvas ecosystem. This initiative aims to resolve the **business problem of inefficiency** in online learning platforms, offering a **scalable, user-friendly tool** that showcases the practical application of language processing technologies in education.

# 2. Design



Our process is divided into seven main parts, and we will introduce each of these seven parts next:

Step 1: User Query Submission

* The user inputs a query (e.g., a question or search request).
* The system captures the query and prepares it for further processing.

Step 2: Query Embedding

* The query is converted into a vector representation using an embedding model.
* The vectorized query is sent to the vector database for similarity matching.

Step 3: Document Retrieval

* The vector warehouse stores document embeddings.
* The system retrieves the top 3 most relevant files based on similarity scores.

Step 4: Text Processing

* Retrieved documents pass through a Text Backbone module to extract relevant sections.
* This module formats the retrieved content into structured data.

Step 5: Query Formatting for GPT-4o

* The system reformats the extracted text into a query-friendly format.
* The structured query is sent to GPT-4o for response generation.

Step 6: GPT-4o(or other LLMs) Answer Generation

* GPT-4o processes the query and extracted content.
* The model generates a contextually relevant and precise answer.

Step 7: Response Delivery

* The system provides the final response to the user.

The response includes:

* Answer to the query
* Source file link & raw text location （for transparency and verification）.

# 3. Scope

## 3.1 Identifying Real-life Data for Language Processing Components

To develop an LLM capable of accurately querying Canvas, the system must collect and structure real-world data from multiple facets of the platform. This data serves as the foundation for the chatbot’s ability to provide context-rich answers.

### Data Sources from Canvas API:

* **Course Information:**  
  Includes course titles, descriptions, schedules, and instructor details.
* **Announcements & Notifications:**  
  Covers instructor updates, system alerts, and event notifications.
* **Assignments & Deadlines:**  
  Contains detailed assignment descriptions, due dates, submission records, and feedback.
* **Discussions & Forums:**  
  Encompasses student and faculty posts, threads, and comments.
* **Grades & Feedback:**  
  Consists of scores, grading rubrics, and instructor remarks.
* **Files & Resources:**  
  Incorporates lecture slides, PDFs, research papers, and supplementary materials.
* **Topic-based Content Retrieval:** Lecture slides will be annotated with key topics (e.g., BERT, transformers, etc.) during data preprocessing. This metadata allows the system to perform precise semantic searches so that, if a user asks, "I'm not sure about BERT; which lecture slide should I refer to?" the chatbot can retrieve the most relevant lecture slides.
* **Quizzes & Exams:**  
  Includes question formats, responses, and performance data.

### Data Collection Strategy:

* **API Integration:**  
  Use the Canvas REST API (or GraphQL API, if available) to fetch structured data securely via OAuth authentication.
* **Local Storage:**  
  Persist retrieved data in a structured database (e.g., PostgreSQL or MongoDB) with scheduled updates to maintain currency.
* **Scalability Considerations:**  
  Address potential rate limits and data synchronization challenges to ensure reliable performance.

## 3.2 Execution Process and Success Measurement Criteria

This section outlines the multi-step process of transforming raw Canvas data into actionable insights while defining key success metrics for system performance.

### Execution Process:

1. **Data Extraction from Canvas API:**
   1. **Authentication:** Secure OAuth-based authentication.
   2. **Retrieval:** Execute REST API calls to fetch data on courses, assignments, announcements, discussions, and lecture slides.
   3. **Storage:** Save data in a structured database for efficient querying.
2. **Data Preprocessing & Embedding Generation:**
   1. **Normalization:** Clean the data (e.g., remove HTML tags and extraneous punctuation) using tools like Spacy or NLTK.
   2. **Embedding Generation:** Convert normalized text into vector representations with models such as LLaMA-2 or DeepSeek.
   3. **Indexing:** Store embeddings in a vector search database (e.g., FAISS or ChromaDB) to support fast similarity searches.
   4. **Metadata Tagging:** For Files & Resources, implement an annotation process to tag lecture slides with key topics to facilitate topic-based queries.
3. **Query Processing with the LLM:**
   1. **User Input:** Capture natural language queries (e.g., “What are my upcoming assignments?” or “Which lecture slide should I refer to for BERT?”).
   2. **Similarity Search & Direct Lookup:** For general queries, retrieve relevant data snippets from indexed content. For topic-specific queries (such as lecture slide recommendations), the system will use the annotated metadata on lecture slides to perform a targeted semantic search.
   3. **Response Generation:** Synthesize the retrieved information into a coherent answer using an LLM (via Hugging Face API or local execution).
4. **User Interaction & Deployment:**
   1. **Interface Development:** Build a user-friendly chatbot interface using PyQt6 (desktop) or Streamlit (web-based).
   2. **Backend Services:** Optionally deploy as a FastAPI service for scalable, cloud-based access with a focus on data privacy.

### Success Measurement Criteria:

* **Accuracy:** Validate that AI-generated responses accurately reflect the underlying Canvas data.
* **Response Relevance:** Evaluate the precision of both general queries and topic-specific content retrieval (e.g., relevant lecture slide recommendations) via human testing and feedback.
* **Latency:** Achieve sub-2-second response times.
* **User Adoption:** Monitor engagement through API request logs.
* **Feedback Integration:** Continuously refine the system based on user feedback from students and instructors.

## 3.3 Suitable Techniques for Each Component

The following table summarizes the techniques and tools chosen to ensure a robust, scalable solution:

|  |  |
| --- | --- |
| **Component** | **Technique & Tools Used** |
| **Data Extraction** | *Technique:* API integration via Canvas REST API with OAuth *Tools:* Canvas API endpoints, secure token management |
| **Text Processing** | *Technique:* Data normalization and cleaning *Tools:* Spacy, NLTK |
| **Vector Embedding** | *Technique:* Dense vector representation generation *Tools:* LLaMA-2 or DeepSeek |
| **Data Storage & Indexing** | *Technique:* Structured data storage and vector search indexing *Tools:* PostgreSQL/MongoDB for data; FAISS/ChromaDB for similarity searches |
| **Files & Resources Retrieval** | *Technique:* Semantic search with topic-based tagging *Tools:* Metadata extraction and tagging tools integrated with FAISS/ChromaDB |
| **Retrieval Augmented Generation (RAG)** | *Technique:* Augmenting responses with retrieved context *Tools:* LangChain, LlamaIndex |
| **LLM Query Processing** | *Technique:* Natural language understanding and synthesis *Tools:* LLaMA-2 or DeepSeek via Hugging Face API or local deployment |
| **User Interface** | *Technique:* Conversational UI design *Tools:* PyQt6 for desktop, Streamlit for web interfaces |
| **Deployment** | *Technique:* Scalable backend setup with privacy in mind *Tools:* FastAPI for backend services, with options for local or cloud-based deployment |

### Major Omissions:

* **Audio Processing:**  
  No support for voice commands.
* **Image Processing:**  
  Does not handle scanned documents or image-based inputs.
* **Assignment Submission:**  
  The system retrieves information only and does not interface with submission APIs.

### Scope Justification:

* **Academic Relevance:** Demonstrates practical application of LLMs for enhanced information retrieval in educational settings.
* **Technical Rigor:** Combines state-of-the-art NLP, embedding techniques, and structured API integration.
* **Real-world Integration:** Offers hands-on experience with data-driven decision making and user interaction design.

# 4. Draft Implementation

The following draft outlines the roadmap for building, testing, and deploying the Canvas Q&A Chatbot.

## 4.1 System Architecture Overview

### Modules:

* **Data Extraction Module:**  
  Integrates with the Canvas API to collect course, assignment, and lecture slide data.
* **Preprocessing & Embedding Module:**  
  Normalizes text, generates vector embeddings for fast similarity search, and annotates lecture slides with topic metadata.
* **Query Processing Module:**  
  Leverages an LLM to synthesize context-aware responses, including both direct data lookups and topic-based recommendations.
* **User Interface Module:**  
  Implements a conversational chatbot interface using PyQt6 or Streamlit.
* **Deployment Module:**  
  Sets up a FastAPI backend for scalable and secure access.

Data Flow Diagram:  
Design a diagram showing data movement from the Canvas API to local storage, through embedding generation and metadata tagging, followed by query processing, and culminating in the user interface.

## 4.2 Development Roadmap

*Phase 1: Data Integration*

* Establish OAuth-based access to the Canvas API.
* Develop scripts to extract essential data (courses, assignments, announcements, lecture slides, etc.).
* Implement local storage using PostgreSQL or MongoDB.

*Phase 2: Data Processing Pipeline*

* Develop a text normalization pipeline using Spacy/NLTK.
* Integrate embedding generation with LLaMA-2 or DeepSeek.
* Set up vector indexing with FAISS or ChromaDB for rapid retrieval.
* **Implement Metadata Tagging:** Automatically annotate lecture slides with key topics during preprocessing to facilitate targeted semantic search.

*Phase 3: Query Processing & LLM Integration*

* Integrate an LLM (e.g., via Hugging Face API or local deployment) for generating responses.
* Implement a retrieval augmented generation (RAG) system using LangChain or LlamaIndex.
* Incorporate semantic search specifically for Files & Resources to support topic-based queries (e.g., for lecture slide recommendations on subjects like BERT).
* Test various query scenarios to ensure proper routing (direct lookup for assignment deadlines versus research queries and topic-based slide retrieval).

*Phase 4: User Interface & Deployment*

* Develop a prototype chatbot UI using PyQt6 (desktop) or Streamlit (web).
* Deploy the backend using FastAPI, ensuring security and scalability.
* Set up logging and monitoring to capture performance metrics and user feedback.

## 4.3 Testing and Validation

* **Unit Testing:** Validate individual modules (data extraction, preprocessing, embedding generation, metadata tagging).
* **Integration Testing:** Ensure smooth data flow and proper functioning of the end-to-end process.
* **User Acceptance Testing (UAT):** Gather and incorporate feedback from target users (students and instructors) to refine both the UI and the accuracy of responses.

## 4.4 Milestones and Success Metrics

* **Minimum Viable Product (MVP):**
  + Basic functionality for data extraction, processing, and direct query response.
  + Achieve sub-2-second response times.
* **Beta Release:**
  + Enhanced query handling with retrieval augmentation and topic-based semantic search for lecture slides, improved accuracy, and initial user feedback integration.
* **Final Release:**
  + Fully integrated system with robust error handling, real-time updates, comprehensive documentation, and live deployment.
  + Ongoing monitoring and iterative improvements based on user feedback.